☐	5878245	March 1999	Johnson et al.	711/154
☐	6014723	January 2000	Tremblay et al.	711/1
☐	6185673	February 2001	Dewan	712/223
☐	6298365	October 2001	Dubey et al.	340/146.2
☐	2002/0029332	March 2002	Saulsbury	712/225

OTHER PUBLICATIONS

Marc Tremblay and William Joy; A Multiple-Thread Processor For Threaded Software Applications: U.S. Ser. No.: 09/204,480; Filed Dec. 3, 1998; 33 pages of Specification (including claims and Abstract); and 10 Sheets of Formal Drawings. (Copy Enclosed).
Linley Gwenap; "MAJC Gives VLIW A New Twist;" Microprocessor Report: The Insider's Guide To Microprocessor Hardware; vol. 13, No. 12; Sep. 13, 1999; 5 pages. (Copy Enclosed).

ART-UNIT: 2183

PRIMARY-EXAMINER: Kim; Kenneth S.

ATTY-AGENT-FIRM: Zagorin, O'Brien & Graham, LLP

ABSTRACT:

The present invention provides a method and apparatus for executing a boundary check instruction that provides accelerated bound checking. The instruction can be used to determine whether an array address represents a null pointer, and whether an array index is less than zero or greater than the size of the array. Three extensions of a boundary check instruction are provided, with each performing a different combination of three boundary check comparisons. One comparison compares a first operand, which may contain the base address of an array, to zero. Another comparison evaluates the value of a second operand, which may contain an index offset, to determine if it is less than zero. The other comparison evaluates whether the value of the second operand is greater than or equal to a third operand. The third operand may indicate the size of an array. A trap is generated if any of the comparisons evaluates to true.

43 Claims, 5 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L9: Entry 19 of 42 File: USPT Mar 9, 2004

US-PAT-NO: 6704862
DOCUMENT-IDENTIFIER: US 6704862 B1

TITLE: Method and apparatus for facilitating exception handling using a conditional trap instruction

DATE-ISSUED: March 9, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Chaudhry; Shailender	San Francisco	CA		
Tremblay; Marc	Menlo Park	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Santa Clara	CA			02

APPL-NO: 09/ 591142 [PALM]
DATE FILED: June 9, 2000

PARENT-CASE:
RELATED APPLICATION This application hereby claims priority under 35 U.S.C. .sctn. 119 to U. S. Provisional Patent Application No. 60/186,979 filed on Mar. 6, 2000.

INT-CL: [07] G06 F 9/38

US-CL-ISSUED: 712/244; 712/24
US-CL-CURRENT: 712/244; 712/24

FIELD-OF-SEARCH: 712/244, 712/24

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5367696	November 1994	Abe	
☐	5634023	May 1997	Adler et al.	
☐	5778219	July 1998	Amerson et al.	
☐	5802337	September 1998	Fielden	

☐	5815719	September 1998	Goebel
☐	5835776	November 1998	Tirumalai et al.
☐	6301705	October 2001	Doshi et al.
☐	6487716	November 2002	Choi et al.

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
WO 99/19795	April 1999	WO	

ART-UNIT: 2183

PRIMARY-EXAMINER: Treat; William M.

ATTY-AGENT-FIRM: Park, Vaughan & Fleming LLP

ABSTRACT:

One embodiment of the present invention provides a system that supports exception handling through use of a conditional trap instruction. The system supports a head thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the head thread. During operation, the system uses the speculative thread to execute code, which includes an instruction that can cause an exception condition. After the instruction is executed, the system determines if the instruction caused the exception condition. If so, the system writes an exception condition indicator to a register. At some time in the future, the system executes a conditional trap instruction which examines a value in the register. If the value in the register is an exception condition indicator, the system executes a trap handling routine to handle the exception condition. Otherwise, the system proceeds with execution of the code. In one embodiment of the present invention, prior to executing the instruction, the system allows a compiler to optimize a program containing the instruction. This optimization process includes scheduling an exception testing instruction associated with the instruction to occupy a free instruction slot following the instruction. This exception testing instruction determines if the instruction causes the exception condition. In one embodiment of the present invention, the trap handling routine triggers a rollback operation to undo operations performed by the speculative thread.

24 Claims, 14 Drawing figures

L9: Entry 19 of 42 File: USPT Mar 9, 2004

US-PAT-NO: 6704862
DOCUMENT-IDENTIFIER: US 6704862 B1

TITLE: Method and apparatus for facilitating exception handling using a conditional trap instruction

DATE-ISSUED: March 9, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Chaudhry; Shailender	San Francisco	CA		
Tremblay; Marc	Menlo Park	CA		

US-CL-CURRENT: 712/244; 712/24

CLAIMS:

What is claimed is:

1. A method for supporting exception handling through use of a conditional trap instruction, the method operating in a system that supports a head thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the head thread, the method comprising: using the speculative thread to execute at least one instruction that performs an operation that can cause an exception condition; determining if the instruction causes the exception condition by executing an exception testing instruction after each of the at least one instructions, wherein the exception testing instruction does not need to immediately follow the tested instruction; if the instruction causes the exception condition, writing an exception condition indicator to a register; and executing the conditional trap instruction which, examines a value in the register, executes a trap handling routine to handle the exception condition if the value in the register is the exception condition indicator, and proceeds with execution of the code if the value in the register is not the exception condition indicator: wherein the head thread executes on a first processor and the speculative thread executes concurrently on a second processor.

2. The method of claim 1, further comprising, prior to executing the at least one instruction, allowing a compiler to optimize a program containing the at least one instruction by scheduling an exception testing instruction associated with each of the at least one instructions to occupy a free instruction slot following the instruction, the exception testing instruction determining if the instruction causes the exception condition.

3. The method of claim 2, further comprising performing register allocation as part of the optimization process.

4. The method of claim 2, wherein the compiler operates as part of a just-in-time compilation mechanism.

5. The method of claim 1, wherein the trap handling routine triggers a rollback operation to undo operations performed by the speculative thread.

6. The method of claim 1, further comprising, prior to executing the instruction, inserting the conditional trap instruction in a vicinity of a return from a code module containing the instruction, so that the conditional trap instruction is executed at the end of the code module, and detects whether instructions within the code module caused an exception condition.

7. The method of claim 1, wherein the at least one instruction includes one of: a getfield instruction that generates the exception condition if a reference to an object containing the field is a null pointer; a putfield instruction that generates the exception condition if a reference to the object containing the field is a null pointer; a check cast instruction that assigns a value to a first variable if the value originates from a second variable with a type that matches the type of the first variable, and if not generates the exception condition; and an array store instruction that assigns a value to an array element if the type of the array element matches the type of a variable from which the value originated, and if not generates the exception condition.

8. The method of claim 1, wherein the conditional trap instruction is located within a very long instruction word (VLIW) instruction.

9. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for supporting exception handling through use of a conditional trap instruction, the method operating in a system that supports a head thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the head thread, the method comprising: using the speculative thread to execute at least one instruction that performs an operation that can cause an exception condition; determining if the instruction causes the exception condition by executing an exception testing instruction after each of the at least one instructions, wherein the exception testing instruction does not need to immediately follow the tested instruction; if the instruction causes the exception condition, writing an exception condition indicator to a register; and executing the conditional trap instruction which, examines a value in the register, executes a trap handling routine to handle the exception condition if the value is the exception condition indicator, and proceeds with execution of the code if the value in the register is not the exception condition indicators; wherein the head thread executes on a first processor and the speculative thread executes concurrently on a second processor.

10. The computer-readable storage medium of claim 9, wherein the method further comprises, prior to executing the at least one instruction, allowing a compiler to optimize a program containing the at least one instruction by scheduling an exception testing instruction associated with each of the at least one instructions to occupy a free instruction slot following the instruction, the exception testing instruction determining if the instruction causes the exception condition.

11. The computer-readable storage medium of claim 10, wherein the method further comprises performing register allocation as part of the optimization process.

12. The computer-readable storage medium of claim 10, wherein the compiler operates as part of a just-in-time compilation mechanism.

13. The computer-readable storage medium of claim 9, wherein the trap handling routine triggers

a rollback operation to undo operations performed by the speculative thread.

14. The computer-readable storage medium of claim 9, wherein the method further comprises, prior to executing the instruction, inserting the conditional trap instruction in a vicinity of a return from a code module containing the instruction, so that the conditional trap instruction is executed at the end of the code module, and detects whether instructions within the code module caused an exception condition.

15. The computer-readable storage medium of claim 9, wherein the at least one instruction includes one of: a getfield instruction that generates the exception condition if a reference to an object containing the field is a null pointer; a putfield instruction that generates the exception condition if a reference to the object containing the field is a null pointer; a check cast instruction that assigns a value to a first variable if the value originates from a second variable with a type that matches the type of the first variable, and if not generates the exception condition; and an array store instruction that assigns a value to an array element if the type of the array element matches the type of a variable from which the value originated, and if not generates the exception condition.

16. The computer-readable storage medium of claim 9, wherein the conditional trap instruction is located within a very long instruction word (VLIW) instruction.

17. An apparatus that supports exception handling through use of a conditional trap instruction, comprising: an operating system that supports a head thread that executes program instructions and a speculative thread that speculatively executes program instructions in advance of the head thread wherein the head thread executes on a first processor and the speculative thread executes concurrently on a second processor; and an execution mechanism that is configured to execute instructions for the speculative thread, the instructions including, at least one instruction that performs an operation that can cause an exception condition, an exception testing instruction, which is placed after each of the at least one instructions, wherein the exception testing instruction does not need to immediately follow the tested instruction; and which, determines if the instruction causes the exception condition, and writes an exception condition indicator to a register if the instruction causes the exception condition, and the conditional trap instruction which, examines a value in the register, executes a trap handling routine to handle the exception condition if the value is the exception condition indicator, and proceeds with execution of the code if the value in the register is not the exception condition indicator.

18. The apparatus of claim 17, further comprising a compiler that optimizes a program containing the at least one instruction by scheduling the exception testing instruction associated with each of the at least one instructions to occupy a free instruction slot following the instruction.

19. The apparatus of claim 18, wherein the compiler is additionally configured to perform register allocation as part of the optimization process.

20. The apparatus of claim 18, wherein the compiler is configured to perform just-in-time compilation.

21. The apparatus of claim 17, wherein the trap handling routine is configured to trigger a rollback operation to undo operations performed by the speculative thread.

22. The apparatus of claim 17, wherein the conditional trap instruction is located in a vicinity of a

return from a code module containing the instruction, so that the conditional trap instruction is executed at the end of the code module, and detects whether instructions within the code module caused an exception condition.

23. The apparatus of claim 17, wherein the at least one instruction includes one of: a getfield instruction that generates the exception condition if a reference to an object containing the field is a null pointer; a putfield instruction that generates the exception condition if a reference to the object containing the field is a null pointer; a check cast instruction that assigns a value to a first variable if the value originates from a second variable with a type that matches the type of the first variable, and if not generates the exception condition; and an array store instruction that assigns a value to an array element if the type of the array element matches the type of a variable from which the value originated, and if not generates the exception condition.

24. The apparatus of claim 17, wherein the conditional trap instruction is located within a very long instruction word (VLIW) instruction.

First Hit Fwd Refs

Generate Collection Print

L12: Entry 2 of 3 File: USPT Apr 1, 2003

US-PAT-NO: 6542990
DOCUMENT-IDENTIFIER: US 6542990 B1

TITLE: Array access boundary check by executing BNDCHK instruction with comparison specifiers

DATE-ISSUED: April 1, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Tremblay; Marc	Menlo Park	CA		
O'Connor; James Michael	Union City	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Santa Clara	CA			02

APPL-NO: 10/ 118388 [PALM]
DATE FILED: April 8, 2002

PARENT-CASE:
CROSS-REFERENCE SECTION This application is a continuation of U.S. patent application Ser. No. 09/565,625, filed May 4, 2000, and entitled, "Array Access Boundary Check By Executing BNDCHK Instruction With Comparison Specifiers," and naming Marc Tremblay and James Michael O'Connor as the inventors, now issued U.S. Pat. No. 6,408,383, the application being incorporated herein by reference in its entirety.

INT-CL: [07] G06 F 12/06

US-CL-ISSUED: 712/227; 711/152, 711/171, 712/208, 712/225
US-CL-CURRENT: 712/227; 711/152, 711/171, 712/208, 712/225

FIELD-OF-SEARCH: 711/152, 711/171, 712/225, 712/227, 712/208

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4760374	July 1988	Moller	340/146.2
☐	5568624	October 1996	Sites et al.	712/223

☐	5878245	March 1999	Johnson et al.	711/154
☐	6014723	January 2000	Tremblay et al.	711/1
☐	6185673	February 2001	Dewan	712/223
☐	6298365	October 2001	Dubey et al.	340/146.2
☐	2002/0029332	March 2002	Saulsbury	712/225

OTHER PUBLICATIONS

Marc Tremblay and William Joy; A Multiple-Thread Processor For Threaded Software Applications: U.S. Ser. No.: 09/204,480; Filed Dec. 3, 1998; 33 pages of Specification (including claims and Abstract); and 10 Sheets of Formal Drawings. (Copy Enclosed).
Linley Gwenap; "MAJC Gives VLIW A New Twist;" Microprocessor Report: The Insider's Guide To Microprocessor Hardware; vol. 13, No. 12; Sep. 13, 1999; 5 pages. (Copy Enclosed).

ART-UNIT: 2183

PRIMARY-EXAMINER: Kim; Kenneth S.

ATTY-AGENT-FIRM: Zagorin, O'Brien & Graham, LLP

ABSTRACT:

The present invention provides a method and apparatus for executing a boundary check instruction that provides accelerated bound checking. The instruction can be used to determine whether an array address represents a null pointer, and whether an array index is less than zero or greater than the size of the array. Three extensions of a boundary check instruction are provided, with each performing a different combination of three boundary check comparisons. One comparison compares a first operand, which may contain the base address of an array, to zero. Another comparison evaluates the value of a second operand, which may contain an index offset, to determine if it is less than zero. The other comparison evaluates whether the value of the second operand is greater than or equal to a third operand. The third operand may indicate the size of an array. A trap is generated if any of the comparisons evaluates to true.

43 Claims, 5 Drawing figures

L12: Entry 2 of 3 File: USPT Apr 1, 2003

DOCUMENT-IDENTIFIER: US 6542990 B1
TITLE: Array access boundary check by executing BNDCHK instruction with comparison specifiers

Parent Case Text (2):
This application is a continuation of U.S. patent application Ser. No. 09/565,625, filed May 4, 2000, and entitled, "Array Access Boundary Check By Executing BNDCHK Instruction With Comparison Specifiers," and naming Marc Tremblay and James Michael O'Connor as the inventors, now issued U.S. Pat. No. 6,408,383, the application being incorporated herein by reference in its entirety.

Brief Summary Text (11):
Moreover, in the context of bounds checking for array accesses, operation of the boundary check instruction accelerates array accesses in which the validity of each array access is checked prior to performing the array access. This robust approach to checking the validity of each array access provides for improved security features, which is desired in various environments, such as a Java.TM. computing environment. For example, this method can be used for various instruction sets such as Sun Microsystems, Inc.'s Majc.TM. instruction set.

Brief Summary Text (14):
When the boundary check instruction is used to perform accelerated bound checking for array accesses, the zero-compare operand is the base address of an array object, the less-than-zero operand is an index offset for an entry in the array, and the upper-range operand is a value indicating the number of entries (i.e., maximum size) in the array.

Brief Summary Text (17):
In another embodiment, executing the boundary check instruction includes performing all three of the comparisons: the zero-compare comparison, the less-than-zero comparison, and the range comparison. For each comparison, a trap is generated if the comparison evaluates to true. When this embodiment is used to perform accelerated bounds checking for array accesses, a trap will be generated if the contents of the zero-compare operand equals zero (i.e., the object pointer for the array to be accessed is a null pointer), if the value of the less-than-zero operand is less than zero (i.e., the index to be accessed is less than zero), or if the value of the less-than-zero operand is greater than or equal to the value of the range operand (i.e., the index to be accessed is greater than or equal to N, where N is the size of the array). The boundary check operation therefore results in a trap prior to the execution of an invalid array access.

Detailed Description Text (7):
FIG. 2 shows a format 202 of a boundary check instruction in accordance with one embodiment of the present invention. The boundary check instruction format 202 includes an 11-bit opcode field 204 in bits <31:21>, which includes an immediate bit field 212. In at least one embodiment, the immediate bit field 212 occupies bit 27 of the opcode field 204. The boundary check instruction format 202 also includes a first register specifier field, rd 206, in bits <20:14>. The rd field 206 contains a zero-compare specifier that indicates the location of a zero-compare operand, where the zero-compare operand is to be compared with zero. When the

BNDCHK instruction is used to perform bounds checking for an array, the zero-compare specifier in the rd field 206 typically indicates a base address of an array. In at least one embodiment, the zero-compare specifier residing in the rd field 206 identifies a particular register. The register identified by the zero-compare specifier in the rd field 206 typically contains the base address of the array to be accessed in a subsequent memory access instruction. In another embodiment, the register identified in the rd field 206 contains a pointer address, with the pointer address containing the base address of the subject array.

Detailed Description Text (8):
FIG. 2 illustrates that the boundary check instruction format 202 also includes a second register specifier field, rs1208, in bits <13:7>. The rs1 field 208 contains a less-than-zero specifier that indicates the location of a less-than-zero operand, where the less-than-zero operand is to be evaluated to determine if its value is less than zero. When the BNDCHK instruction is used to perform array bounds checking, the less-than-zero operand is typically an array index offset. In at least one embodiment, the less-than-zero specifier residing in the rs1 field 208 identifies a particular register. The register identified by the less-than-zero specifier in the rs1 field 208 typically contains an offset from the base address of the array to be accessed in a subsequent memory access instruction, the offset identifying a particular entry of the array to be accessed. In another embodiment, the register identified in the rs1 field 208 contains a pointer address, with the pointer address containing the index offset for the array entry.

Detailed Description Text (9):
FIG. 2 illustrates that the boundary check instruction format 202 also includes a third register/immediate specifier field 210 in bits <6:0>. The third register/immediate specifier field 210 contains an upper range specifier. The type of data in the upper range specifier corresponds to the value of the contents of the immediate bit field 212. If the value of the immediate bit ("i-bit") residing in the immediate bit field 212 is set to binary value of 1b'1', (i.e., a logic-high state) then the upper range specifier contains a sign-extended immediate value (simm7) to be compared with the contents of the register specified by rs1208. In contrast, if the value of the i-bit in the immediate bit field 212 is reset to a binary value of 1b'0' (i.e., a logic-low state), then the upper range specifier contains a range-operand specifier that identifies the location of an upper-range operand, where the upper-range operand is to be compared with the contents of the register specified by rs1. When the BNDCHK instruction is used to perform accelerated bound checking for array accesses, the range-operand specifier residing in the rs2 field 210 identifies a particular register. The register identified by the range-operand specifier in the rs2 field 210 typically contains a numerical value indicating the number of entries in the array to be accessed. For instance, for an array of N entries, indexed from 0 to N-1, the value in the register specified by the range-operand specifier in the rs2 field 210 is N. In another embodiment, the register identified in the rs2 field 210 contains a pointer address, with the pointer address containing a value indicating the number of entries in the subject array.

Detailed Description Text (15):
In a first extension of the BNDCHK instruction, the Zero Compare, the Less-than-zero Comparison, and the Range Comparisons are performed, and a trap is generated if any of the three Comparisons evaluate to true. The first extension is particularly useful when performing accelerated bound checking for array accesses, discussed in further detail below.

Detailed Description Text (17):
A third extension of the BNDCHK instruction performs the Zero Compare and the Less-than-zero Comparison (but not the Range Comparison). This third extension is useful for performing a less robust array bound checking function. In the context of array accesses, the third extension of BNDCHK will generate a trap when the value of R

[rd] is zero, indicating a null pointer as the base address for the array, OR when the value of R[rs1] is less than zero, indicating an invalid array index value.

Detailed Description Text (18):
FIG. 3 is a functional diagram of one embodiment of the operation of the boundary check instruction. In particular, FIG. 3 illustrates the operation of the first extension of the BNDCHK instruction, wherein the Zero-Compare, Less-than-zero, and Range Comparisons are performed. The discussion of FIG. 3 will assume, for ease of discussion, that the BNDCHK instruction illustrated in FIG. 3 is being used to perform accelerated bound checking for an array access. One skilled in the art will recognize that the second and third extensions of the BNDCHK instruction implement only selected portions of the functionality discussed herein, and that any of the extensions may be used for bound-checking in contexts other than array accesses.

Detailed Description Text (19):
In at least one embodiment, the boundary check instruction ("BNDCHK") is implemented in a microprocessor that performs array operations. BNDCHK accelerates a check of the validity of an array access by eliminating two (second and third extensions) or three (first extension) conditional branches for valid array accesses. The boundary check instruction can be used in a computing environment in which security features like checking the validity of an array access prior to performing the array access are desired, such as in a Java.TM. computing environment.

Detailed Description Text (21):
The present invention solves this problem by providing a boundary check instruction that can be used, in the context of array bound-checking, to perform various combinations of the following comparisons: Compare the base address of the subject array with zero Determine whether the index offset is less than zero, and Determine whether the index offset is greater than N-1, where N is the size of the array.

Detailed Description Text (22):
If any of these conditions checked by the boundary check instruction are met, then the boundary check instruction automatically traps. Otherwise, execution continues uninterrupted. The first extension of the boundary check operation eliminates three conditional branches for an array access and therefore allows for better code motion through the microprocessor.

Detailed Description Text (23):
FIG. 3 illustrates that BNDCHK 302, which includes opcode 304, is a control flow instruction that causes a trap if any of the Comparisons evaluates to true. Regarding the Zero Compare, the BNDCHK instruction 302 causes a trap if the value in the register specified by rd 206, is equal zero. FIG. 3 illustrates that, when the BNDCHK instruction is used to perform boundary check acceleration for array accesses, the register specified in the rd field 206 contains the array object pointer 306. Accordingly, the BNDCHK instruction traps if the array object pointer 306 is null (i.e., equal to zero) and therefore does not point to a valid array address.

Detailed Description Text (24):
Regarding the Less-than-zero Comparison, the BNDCHK instruction also causes a trap if the value residing in the register specified by the rs1 field 208 is less than zero. When using the BNDCHK instruction to perform boundary check acceleration for array accesses, R[rs1] contains the array index offset 308 for target array entry 314 of array 312. Accordingly, the BNDCHK instruction causes a trap if the value of the array index offset 308 is less than zero and therefore represents an invalid index value.

Detailed Description Text (25):
Regarding the Range Comparison, the BNDCHK instruction 302 also causes a trap if

the array index offset value 308 is greater than or equal to the upper range value 310. As is described above, the upper range value resides in the register specified by the range-operand specifier (rs2) in the third register/immediate specifier field 210 when the i-bit is reset. When the i-bit is set, the upper range value is the immediate value simm7 residing in the third register/immediate specifier field 210. When the BNDCHK instruction 302 is used to perform boundary check acceleration for array accesses, the upper range value 310 is the number of array entries (N) in array 312. In order for a valid array access to occur, the array index offset value 308 must be less than N, assuming that the initial array entry has an index offset value of zero.

US Reference Patent Number (4):
6014723

L12: Entry 2 of 3 File: USPT Apr 1, 2003

US-PAT-NO: 6542990
DOCUMENT-IDENTIFIER: US 6542990 B1

TITLE: Array access boundary check by executing BNDCHK instruction with comparison specifiers

DATE-ISSUED: April 1, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Tremblay; Marc	Menlo Park	CA		
O'Connor; James Michael	Union City	CA		

US-CL-CURRENT: 712/227; 711/152, 711/171, 712/208, 712/225

CLAIMS:

What is claimed is:

1. An apparatus that performs for boundary check acceleration, comprising: means for executing a boundary check instruction, (to facilitate boundary check acceleration) the boundary check instruction including an opcode and also including a less-than-zero specifier, the less-than-zero specifier indicating the location of a less-than-zero operand; wherein means for executing the boundary check instruction further comprises means for performing a less-than-zero comparison wherein it is determined whether the less-than-zero operand is less than zero; and wherein means for executing the boundary check instruction further comprises one or more additional means for performing comparison, the plurality of additional means for performing comparison including: means for determining whether a zero-compare operand is equal to zero; and means for determining whether the value of the less-than-zero operand is greater than or equal to the value of an upper-range operand.

2. The apparatus of claim 1, further comprising: means for decoding the boundary check instruction.

3. The apparatus of claim 1, further comprising: means for generating a trap if the value of the less-than-zero operand is less than zero.

4. The apparatus of claim 1 wherein the less-than-zero operand comprises a base address of an array.

5. The apparatus of claim 1 wherein: the opcode includes an immediate bit; the boundary check instruction further includes means for indicating, if the immediate bit is in a first state, the location of the upper-range operand, and means for indicating if the immediate bit is in a second state, the upper-range operand; and the one or more additional means further comprises means for

determining whether the less-than-zero operand is greater than or equal to the value of upper-range operand.

6. The apparatus of claim 5 wherein the upper-range operand comprises, if the immediate bit is in the second state, an immediate value.

7. The apparatus of claim 5 wherein the second state comprises a logic-high state.

8. The apparatus of claim 5 further comprising: means for decoding the boundary check instruction.

9. The apparatus of claim 5 wherein: the value of the upper-range operand indicates the size of an array; and the less-than-zero operand comprises an index offset for an entry in the array.

10. The apparatus of claim 5 further comprising: means for generating a trap if the value of the less-than-zero operand is less than zero.

11. The apparatus of claim 5 further comprising: means for generating a trap if the value of the less-than-zero operand is greater than or equal to the value of the upper-range operand.

12. The apparatus of claim 1 wherein: the boundary check instruction further includes a zero-compare specifier, the zero-compare specifier identifying the location of the zero-compare operand; and the one or more additional means comprises means for determining whether the zero-compare operand is equal to zero.

13. The apparatus of claim 12 further comprising: means for generating a trap if the value of the zero-compare operand is equal to zero.

14. The apparatus of claim 12 further comprising: means for decoding the boundary check instruction.

15. The apparatus of claim 12 further comprising: means for generating a trap if the value of the less-than-zero operand is less than zero.

16. The apparatus of claim 12 wherein: the zero-compare operand comprises a base address of an array; and the less-than-zero operand comprises an index offset for an entry in the array.

17. The apparatus of claim 5 wherein: the boundary check instruction further includes means for identifying the location of the zero-compare operand; and the one or more additional means further comprises means for determining whether the less-than-zero operand is greater than or equal to the value of the upper-range operand and means for determining whether the value of the less-than-zero operand is greater than or equal to the value of the upper-range operand.

18. The apparatus of claim 17 wherein the upper-range operand comprises, if the immediate bit is in the second state, an immediate value.

19. The apparatus of claim 17, further comprising: means for decoding the boundary check instruction.

20. The apparatus of claim 17, further comprising: means for generating a trap if the value of the less-than-zero operand is less than zero.

21. The apparatus of claim 17, further comprising: means for generating a trap if the value of the zero-compare operand is equal to zero.

22. The apparatus of claim 17, further comprising: means for generating a trap if the value of the less-than-zero operand is greater than or equal to the value of the upper-range operand.

23. The apparatus of claim 17, wherein: the zero-compare operand comprises a base address of an array; the less-than-zero operand comprises an index offset for an entry in the array; and the value of the upper-range operand indicates the size of the array.

24. An apparatus that executes a boundary check instruction including a less-than-zero specifier configured to indicate a location of a less-than-zero operand, the apparatus comprising: a comparator configured to perform a less-than-zero comparison whereby it is determined whether the less-than-zero operand is less than zero; the comparator further being configured to perform one or more additional comparisons from a set of: a zero-compare comparison whereby it is determined whether a zero-compare operand is equal to zero; and a range comparison whereby it is determined whether the value of the less-than-zero operand is greater than or equal to the value of an upper-range operand.

25. The apparatus of claim 24, further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is less than zero.

26. The apparatus of claim 24 wherein the less-than-zero operand comprises a base address of an array.

27. The apparatus of claim 24 wherein: the boundary check instruction further being configured to include an opcode including an immediate bit; an upper-range specifier, the upper-range specifier being configured to indicate, if the immediate bit is in a first state, the location of the upper-range operand, the upper-range specifier being configured to indicate, if the immediate bit is in a second state, the upper-range operand.

28. The apparatus of claim 27 wherein the upper-range operand is configured to comprise, if the immediate bit is in the second state, an immediate value.

29. The apparatus of claim 27 wherein the second state comprises a logic-high state.

30. The apparatus of claim 27 wherein: the value of the upper-range operand indicates the size of an array; and the less-than-zero operand comprises an index offset for an entry in the array.

31. The apparatus of claim 27 further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is less than zero.

32. The apparatus of claim 27 further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is greater than or equal to the value of the upper-range operand.

33. The apparatus of claim 24 wherein: the boundary check instruction is further configured to include a zero-compare specifier, the zero-compare specifier being configured to identify the location of the zero-compare operand; and the comparator further being configured to perform the zero-compare comparison.

34. The apparatus of claim 33 further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the zero-compare operand is equal to zero.

35. The apparatus of claim 33 further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is less than zero.

36. The apparatus of claim 33 wherein: the zero-compare operand comprises a base address of an array; and the less-than-zero operand comprises an index offset for an entry in the array.

37. The apparatus of claim 27 wherein: the boundary check instruction is further configured to include a zero-compare specifier, the zero-compare specifier being configured to identify the location of the zero-compare operand; and the comparator further being configured to perform the zero-compare comparison and the range comparison.

38. The apparatus of claim 37 wherein the upper-range operand being configured to comprise, if the immediate bit is in the second state, an immediate value.

39. The apparatus of claim 37 wherein the second state comprises a logic-high state.

40. The apparatus of claim 37, further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is less than zero.

41. The apparatus of claim 37, further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the zero-compare operand is equal to zero.

42. The apparatus of claim 37, further comprising: a trap module coupled to the comparator, the trap module being configured to generate a trap if the value of the less-than-zero operand is greater than or equal to the value of the upper-range operand.

43. The apparatus of claim 37, wherein: the zero-compare operand comprises a base address of an array; the less-than-zero operand comprises an index offset for an entry in the array; and the value of the upper-range operand indicates the size of the array.

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, April 27, 2004

Hide?	Set Name	Query	Hit Count
	DB=USPT; PLUR=NO; OP=ADJ		
☐	L12	l2 and L11	3
☐	L11	array with bound$5	6125
☐	L10	6014723.PN.	1
	DB=DWPI; PLUR=NO; OP=ADJ		
☐	L9	trap and L8	3
☐	L8	chaudhry.in. and tremblay.in.	32
	DB=EPAB; PLUR=NO; OP=ADJ		
☐	L7	chaudhry.in. and tremblay.in.	10
	DB=PGPB,USPT; PLUR=NO; OP=ADJ		
☐	L6	trap instruction with zero	42
☐	L5	trap adj zero	14
☐	L4	trap with zero	807
☐	L3	multiple bit boundary value	0
☐	L2	6014723.uref.	9
☐	L1	bartkowiak.in. and advanced.asn.	40

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection Print

L15: Entry 5 of 18 File: USPT Feb 4, 2003

US-PAT-NO: 6516405
DOCUMENT-IDENTIFIER: US 6516405 B1

TITLE: Method and system for safe data dependency collapsing based on control-flow speculation

DATE-ISSUED: February 4, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Jourdan; Stephan J.	Portland	OR		
Gabbay; Freddy	Tel-Aviv			IL
Ronen; Ronny	Haifa			IL
Yoaz; Adi	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Intel Corporation	Santa Clara	CA			02

APPL-NO: 09/ 475646 [PALM]
DATE FILED: December 30, 1999

INT-CL: [07] G06 F 9/45

US-CL-ISSUED: 712/216; 712/226
US-CL-CURRENT: 712/216; 712/226

FIELD-OF-SEARCH: 712/216, 712/226

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5452426	September 1995	Papworth et al.	712/217
☐	5784604	July 1998	Muhich et al.	712/238
☐	5819057	October 1998	Witt et al.	712/204
☐	5835744	November 1998	Tran et al.	711/214
☐	5919256	July 1999	Widigen et al.	711/100
	6000028	December 1999	Chernoff et al.	712/226

☐				
☐	6223254	April 2001	Soni	711/123
☐	6260190	July 2001	Ju	712/205

OTHER PUBLICATIONS

U.S. patent application Ser. No. 09/348,403, Jourdan et al., filed Jul. 7, 1999.*
U.S. patent application Ser. No. 09/348,404, Jourdan et al., filed Jul. 7, 1999.*
U.S. patent application Ser. No. 09/348,973, Jourdan et al., filed Jul. 7, 1999.

ART-UNIT: 2783

PRIMARY-EXAMINER: Coleman; Eric

ATTY-AGENT-FIRM: Kenyon & Kenyon

ABSTRACT:

The present invention is directed to an apparatus and method for data collapsing based on control-flow speculation (conditional branch predictions). Because conditional branch outcomes are resolved based on actual data values, the conditional branch prediction provides potentially valuable insight into data values. Upon encountering a branch if equal instruction and this instruction is predicted as taken or a branch if not equal instruction and this instruction is predicted as not taken, this invention assumes that the two operands used to determine the conditional branch are equal. The data predictions are safe because a data misprediction means a conditional branch misprediction which results in a pipeline flush of the instructions following the conditional branch instruction including the data mispredictions.

38 Claims, 13 Drawing figures

L15: Entry 5 of 18 File: USPT Feb 4, 2003

DOCUMENT-IDENTIFIER: US 6516405 B1
TITLE: Method and system for safe data dependency collapsing based on control-flow speculation

Detailed Description Text (11):
The compare operation and the setting of the zero flag, however, may be performed by other instructions which are not an explicit compare instruction. These instructions which modify the zero flag but are not explicit compare instructions will be referred to hereinafter as implicit compare instructions. These implicit compare instructions compare the destination register with zero after executing the instruction. For example, from the perspective of the zero flag, the instruction "add eax, 4" is the same as the instructions "add eax, 4" and "cmp eax, 0". Because this invention may be practiced using either an explicit compare instruction or an implicit compare instruction, we will not differentiate between the two and will simply refer to these instructions that modify the zero flag as compare instructions.

First Hit Fwd Refs

Generate Collection Print

L15: Entry 6 of 18 File: USPT May 28, 2002

US-PAT-NO: 6397239
DOCUMENT-IDENTIFIER: US 6397239 B2

TITLE: Floating point addition pipeline including extreme value, comparison and accumulate functions

DATE-ISSUED: May 28, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Oberman; Stuart F.	Sunnyvale	CA		
Juffa; Norbert	San Jose	CA		
Weber; Fred	San Jose	CA		
Ramani; Krishnan	Sunnyvale	CA		
Cherukuri; Ravi Krishna	Milpitas	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 09/ 778352 [PALM]
DATE FILED: February 6, 2001

PARENT-CASE:
This application is a divisional application of U.S. patent application Ser. No. 09/055,916, filed Apr. 6, 1998 now U.S. Pat. No. 6,298,367.

INT-CL: [07] G06 F 7/42, G06 F 7/38

US-CL-ISSUED: 708/505; 708/495
US-CL-CURRENT: 708/505; 708/495

FIELD-OF-SEARCH: 708/505, 708/204, 708/490, 708/495

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4644466	February 1987	Saito	
☐	5136536	August 1992	Ng	708/490

☐	5369607	November 1994	Okamoto	708/490
☐	5467476	November 1995	Kawasaki	
☐	5483476	January 1996	Horen et al.	708/490
☐	5515306	May 1996	Blaner et al.	
☐	5561615	October 1996	Kuo et al.	
☐	5568412	October 1996	Han et al.	708/490
☐	5619198	April 1997	Blackham et al.	
☐	5715186	February 1998	Curtet	
☐	5732007	March 1998	Grushin et al.	708/490
☐	5764548	June 1998	Keith et al.	
☐	5764556	June 1998	Stiles	
☐	5790445	August 1998	Eisen et al.	
☐	5808926	September 1998	Gorshtein et al.	708/490
☐	5859997	January 1999	Peleg et al.	708/490
☐	5954790	September 1999	Wong	
☐	5963461	October 1999	Gorshtein et al.	

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0 463 975	January 1992	EP	
0 678 808	October 1995	EP	
94/18632	August 1994	WO	
96/17292	June 1996	WO	

OTHER PUBLICATIONS

IBM Technical Disclosure, "ALU Implementing Native Minimum/Maximum Function for Signal Processing Applications," Oct. 1986.
Schulte et al.," Symmetric Bipartite Tables for Accurate Functions Approximation, "1997, pp. 175-183.
Hassler et al., "Function Evaluation by Table Look-up and Addition, "1995, pp. 10-16.
Das Sarma et al., "Faithful Bipartite ROM Reciprocal Tables," 1995, pp. 17-28.

ART-UNIT: 2121

PRIMARY-EXAMINER: Mai; Tan V.

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kivlin; B. Noel

ABSTRACT:

A multimedia execution unit configured to perform vectored floating point and integer instructions. The execution unit may include an add/subtract pipeline having far and close data paths. The far path is configured to handle effective

addition operations and effective subtraction operations for operands having an absolute exponent difference greater than one. The close path is configured to handle effective subtraction operations for operands having an absolute exponent difference less than or equal to one. The close path is configured to generate two output values, wherein one output value is the first input operand plus an inverted version of the second input operand, while the second output value is equal to the first output value plus one. Selection of the first or second output value in the close path effectuates the round-to-nearest operation for the output of the adder.

56 Claims, 88 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L15: Entry 6 of 18 File: USPT May 28, 2002

US-PAT-NO: 6397239
DOCUMENT-IDENTIFIER: US 6397239 B2

TITLE: Floating point addition pipeline including extreme value, comparison and accumulate functions

DATE-ISSUED: May 28, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Oberman; Stuart F.	Sunnyvale	CA		
Juffa; Norbert	San Jose	CA		
Weber; Fred	San Jose	CA		
Ramani; Krishnan	Sunnyvale	CA		
Cherukuri; Ravi Krishna	Milpitas	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 09/ 778352 [PALM]
DATE FILED: February 6, 2001

PARENT-CASE:
This application is a divisional application of U.S. patent application Ser. No. 09/055,916, filed Apr. 6, 1998 now U.S. Pat. No. 6,298,367.

INT-CL: [07] G06 F 7/42, G06 F 7/38

US-CL-ISSUED: 708/505; 708/495
US-CL-CURRENT: 708/505; 708/495

FIELD-OF-SEARCH: 708/505, 708/204, 708/490, 708/495

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4644466	February 1987	Saito	
☐	5136536	August 1992	Ng	708/490

☐	5369607	November 1994	Okamoto	708/490
☐	5467476	November 1995	Kawasaki	
☐	5483476	January 1996	Horen et al.	708/490
☐	5515306	May 1996	Blaner et al.	
☐	5561615	October 1996	Kuo et al.	
☐	5568412	October 1996	Han et al.	708/490
☐	5619198	April 1997	Blackham et al.	
☐	5715186	February 1998	Curtet	
☐	5732007	March 1998	Grushin et al.	708/490
☐	5764548	June 1998	Keith et al.	
☐	5764556	June 1998	Stiles	
☐	5790445	August 1998	Eisen et al.	
☐	5808926	September 1998	Gorshtein et al.	708/490
☐	5859997	January 1999	Peleg et al.	708/490
☐	5954790	September 1999	Wong	
☐	5963461	October 1999	Gorshtein et al.	

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0 463 975	January 1992	EP	
0 678 808	October 1995	EP	
94/18632	August 1994	WO	
96/17292	June 1996	WO	

OTHER PUBLICATIONS

IBM Technical Disclosure, "ALU Implementing Native Minimum/Maximum Function for Signal Processing Applications," Oct. 1986.
Schulte et al.," Symmetric Bipartite Tables for Accurate Functions Approximation, "1997, pp. 175-183.
Hassler et al., "Function Evaluation by Table Look-up and Addition, "1995, pp. 10-16.
Das Sarma et al., "Faithful Bipartite ROM Reciprocal Tables," 1995, pp. 17-28.

ART-UNIT: 2121

PRIMARY-EXAMINER: Mai; Tan V.

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kivlin; B. Noel

ABSTRACT:

A multimedia execution unit configured to perform vectored floating point and integer instructions. The execution unit may include an add/subtract pipeline having far and close data paths. The far path is configured to handle effective

addition operations and effective subtraction operations for operands having an absolute exponent difference greater than one. The close path is configured to handle effective subtraction operations for operands having an absolute exponent difference less than or equal to one. The close path is configured to generate two output values, wherein one output value is the first input operand plus an inverted version of the second input operand, while the second output value is equal to the first output value plus one. Selection of the first or second output value in the close path effectuates the round-to-nearest operation for the output of the adder.

56 Claims, 88 Drawing figures

L15: Entry 6 of 18 File: USPT May 28, 2002

DOCUMENT-IDENTIFIER: US 6397239 B2
TITLE: Floating point addition pipeline including extreme value, comparison and accumulate functions

Detailed Description Text (176):
The vectored floating point instructions described above are particularly useful in the geometry processing stages of a 3-D graphics pipeline. Another class of functions commonly utilized in graphics processing are extreme value functions. As used herein, "extreme value functions" are those functions which return as a result either a maximum or minimum value selected among a plurality of values. In typical multimedia systems, a minimum value or a maximum value is obtained through the execution of several sequentially executed instructions. For example, a compare instruction may first be executed to determine the relative magnitudes of a pair of operand values, and subsequently a conditional branch instruction may be executed to determine whether a move operation must be performed to move the extreme value to a destination register or other storage location. These sequences of commands commonly occur in multimedia applications, such as in clipping algorithms for graphics rendering systems. Since extreme value functions are implemented through the execution of multiple instructions, however, a relatively large amount of processing time may be consumed by such operations. Graphics processing efficiency may be advantageously increased by dedicated extreme value instructions as described below with reference to FIGS. 45-46.

First Hit Fwd Refs

Generate Collection Print

L15: Entry 15 of 18 File: USPT Feb 22, 2000

US-PAT-NO: 6029244
DOCUMENT-IDENTIFIER: US 6029244 A

TITLE: Microprocessor including an efficient implementation of extreme value instructions

DATE-ISSUED: February 22, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Oberman; Stuart	Sunnyvale	CA		
Juffa; Norbert	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 08/ 948679 [PALM]
DATE FILED: October 10, 1997

PARENT-CASE:
PRIORITY CLAIM This application claims the benefit of U.S. Provisional Application No. 60/063,601, entitled Method and Apparatus for Multifunction Arithmetic, filed Oct. 23, 1997.

INT-CL: [07] G06 F 9/305

US-CL-ISSUED: 712/223
US-CL-CURRENT: 712/223

FIELD-OF-SEARCH: 712/223, 712/208

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5515306	May 1996	Blaner et al.	
☐	5560032	September 1996	Nguyen	395/800
☐	5715186	February 1998	Curtet	
☐	5867683	February 1999	Witt et al.	395/394

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0463975A2	January 1992	EP	
0678808A1	October 1995	EP	
96/17292	June 1996	WO	

OTHER PUBLICATIONS

International Search Report for PCT/US 98/12666 dated Oct. 12, 1998.
IBM Technical Disclosure, "ALU Implementing Hative Miniumum/Maximum Functioin For Signal Processing Applications," vol. 29, No. 5, Oct. 1986.
English language abstract as downloaded and printed from Dialog Website for EPO0678808A1, 1 page, from priority application date Apr. 17, 1995.

ART-UNIT: 273

PRIMARY-EXAMINER: Eng; David Y.

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kivlin; B. Noel

ABSTRACT:

An execution unit is provided for executing a first instruction which includes an opcode field, a first operand field, and a second operand field. The execution unit includes a first input register for receiving a first operand specified by a value of the first operand field, and a second input register for receiving a second operand specified by a value of the second operand field. The execution unit further includes a comparator unit which is coupled to receive a value of the opcode field for the first instruction. The comparator unit is also coupled to receive the first and second operand values from the first and second input registers, respectively. The execution further includes a multiplexer which receives a plurality of inputs. These inputs include a first constant value, a second constant value, and the values of the first and second operand. If the decoded opcode value received by the comparator indicates that the first instruction is either a compare or extreme value function, the comparator conveys one or more control signals to the multiplexer for the purpose of selecting an output of the multiplexer as the result of the first instruction. If the first instruction is one of a plurality of extreme value instructions, the one or more control signals conveyed by the comparator unit select between the first operand and second operand to determine the result of the first instruction. If the first instruction is one of a plurality of compare instructions, the one or more control signals conveyed by the comparator unit select between the first and second constant value to determine the result of the first instruction. In another embodiment, a similar execution unit is provided which handles vector operands.

45 Claims, 20 Drawing figures

L15: Entry 15 of 18 File: USPT Feb 22, 2000

DOCUMENT-IDENTIFIER: US 6029244 A
TITLE: Microprocessor including an efficient implementation of extreme value instructions

Brief Summary Text (7):
Another set of functions commonly utilized in multimedia processing are the extreme value functions. As used herein, "extreme value functions" are those functions which return either a minimum value selected among a plurality of values, or a maximum value selected among a plurality of values as a result of the function. In typical multimedia systems, a minimum value or a maximum value is obtained through the execution of several sequentially executed instructions. For example, a compare instruction may first be executed to determine the relative magnitudes of a pair of operand values, and subsequently a conditional branch instruction may be executed to determine whether a move operation must be performed to move the extreme value to a destination register or other storage location. These sequences of commands are common in multimedia applications, such as clipping algorithms in graphics rendering systems. Since extreme value functions are implemented through the execution of multiple instructions, however, a relatively large amount of processing time may be consumed by such operations.